

# COMP 122/L Lecture 12

Kyle Dewey

# Outline

- Memory instructions
  - Load (`ldr`)
  - Store (`str`)
- Arrays

# Memory Operations

# Refresher

You've already seen one form of `ldr` for handling strings

# Refresher

You've already seen one form of `ldr` for handling strings

---

```
.data
my_string:
    .asciz "hello"

.text
ldr r0, =my_string
```

# Refresher

You've already seen one form of `ldr` for handling strings

---

```
.data  
my_string:  
    .asciz "hello"
```

```
.text  
ldr r0, =my_string
```

Puts starting address of "hello" in r0

# Putting Integers in Memory

`.word` directive will put a 32-bit integer in memory,  
much like `.asciz` will put a string in memory

# Putting Integers in Memory

`.word` directive will put a 32-bit integer in memory,  
much like `.asciz` will put a string in memory

---

```
        .data
my_string:
        .asciz "hello"
first_int:
        .word 42
second_int:
        .word 38
```



# Reading Integers From Memory

Step 1: use `ldr` to put its address into a register..

# Reading Integers From Memory

Step 1: use `ldr` to put its address into a register..

---

```
.data
first_int:
    .word 42
second_int:
    .word 38

.text
ldr r0, =first_int
```

# Reading Integers From Memory

Step 2: use `ldr` with `[]` to read the value at the address

---

```
.data
first_int:
    .word 42
second_int:
    .word 38

.text
ldr r0, =first_int
```

# Reading Integers From Memory

Step 2: use `ldr` with `[]` to read the value at the address

---

```
.data
first_int:
    .word 42
second_int:
    .word 38

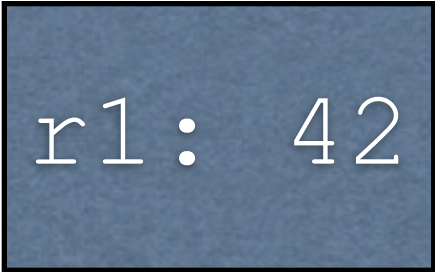
.text
ldr r0, =first_int
ldr r1, [r0]
```

# Reading Integers From Memory

Step 2: use `ldr` with `[]` to read the value at the address

```
.data
first_int:
    .word 42
second_int:
    .word 38
```

```
.text
ldr r0, =first_int
ldr r1, [r0]
```



r1: 42

# Writing Integers to Memory

Step 1: use `ldr` to put its address into a register...

# Writing Integers to Memory

Step 1: use `ldr` to put its address into a register...

---

```
.data
first_int:
    .word 42
second_int:
    .word 38

.text
ldr r0, =first_int
```

# Writing Integers to Memory

Step 2: use `str` to write a value at that address

---

```
        .data
first_int:
        .word 42
second_int:
        .word 38

        .text
ldr r0, =first_int
```



# Writing Integers to Memory

Step 2: use `str` to write a value at that address

```
.data
first_int:
    .word 42 57
second_int:
    .word 38

.text
ldr r0, =first_int
mov r1, #57
str r1, [r0]
```

**Example:**

memory\_variables.s

# Arrays

# Specifying Arrays

Only distinction from variables:

multiple values are specified with the `.word` directive

# Specifying Arrays

Only distinction from variables:

multiple values are specified with the `.word` directive

---

```
.data  
first_int:  
    .word 42
```

# Specifying Arrays

Only distinction from variables:

multiple values are specified with the `.word` directive

---

```
.data
first_int:
    .word 42
array:
    .word 32, 65, 76, 87
```

# Accessing Arrays

Basic approach: increment memory address

# Accessing Arrays

Basic approach: increment memory address

---

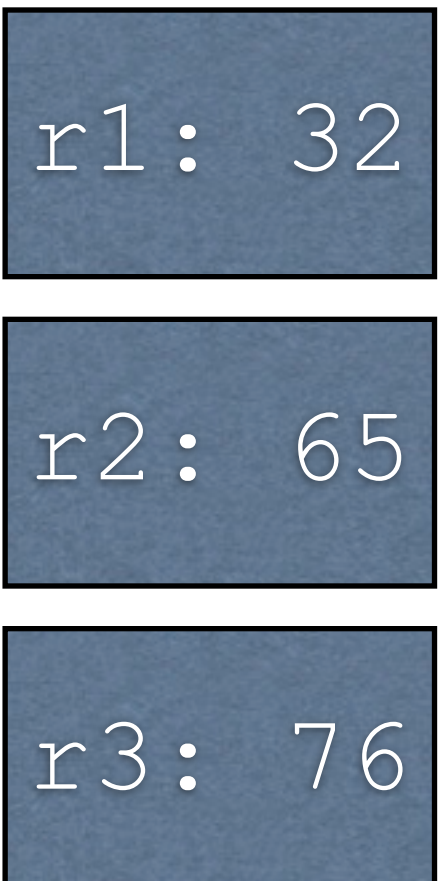
```
.data
arr:
    .word 32, 65, 76
    .text
ldr r0, =arr
ldr r1, [r0]
add r0, r0, #4
ldr r2, [r0]
add r0, r0, #4
ldr r3, [r0]
```



# Accessing Arrays

Basic approach: increment memory address

```
.data
arr:
    .word 32, 65, 76
    .text
ldr r0, =arr
ldr r1, [r0]
add r0, r0, #4
ldr r2, [r0]
add r0, r0, #4
ldr r3, [r0]
```



r1: 32
r2: 65
r3: 76

# Accessing Arrays

Basic approach: increment memory address

<code>.data</code>	32	65	76
<code>arr:</code>	<code>arr</code>	<code>arr + 4</code>	<code>arr + 8</code>
<code>.word 32, 65, 76</code>			
<code>.text</code>			
<code>ldr r0, =arr</code>	<code>r1: 32</code>		
<code>ldr r1, [r0]</code>			
<code>add r0, r0, #4</code>	<code>r2: 65</code>		
<code>ldr r2, [r0]</code>			
<code>add r0, r0, #4</code>			
<code>ldr r3, [r0]</code>	<code>r3: 76</code>		

**Example:**

register\_indirect.s

# More on Memory Access

```
ldr r3, [r0]
```

- The above instruction uses the *register indirect addressing mode*
  - Addressing mode: how the processor accesses something
  - Register indirect: Memory access is done through an address in a register
- Many more available: see `register_*.s`

## **Array Access Example:**

```
print_array_fixed_length.s
```

# Writing to Array Example:

```
write_array_increasing.s
```

## Another Array Access Example:

```
print_array_variable_length.s
```